4

Trees

4.1 Characterizations of Trees

A *tree* is a connected graph that contains no cycle. Figure 4.1 contains three examples of trees. It is also clear that every path is a tree, and the star $K_{1,n}$ is a tree for every n.

A tree is a minimal connected graph in the following sense: if any vertex of degree at least 2, or any edge, is deleted, then the resulting graph is not connected. In fact it is easy to prove the following stronger theorem; the proof is left as an exercise.

Theorem 4.1. *A connected graph is a tree if and only if every edge is a bridge.*

Trees are also characterized among connected graphs by their number of edges.

Theorem 4.2. *A finite connected graph G with v vertices is a tree if and only if it has exactly $v - 1$ edges.*

Proof. (i) Suppose that G is a tree with v vertices. We proceed by induction on v. The theorem is true for $v = 1$, since the only graph with one vertex is K_1, which is a tree. Suppose it is true for $w < v$, and suppose G is a tree with v vertices. Select an edge (G must have an edge, or it will be the unconnected graph $\bar{K}_v$) and delete it. The result is a union of two disjoint components, each of which is a tree with less than v vertices; say the first component has v_1 vertices and the second has v_2, where $v_1 + v_2 = v$. By the induction hypothesis, these graphs have $v_1 - 1$ and $v_2 - 1$ edges respectively.

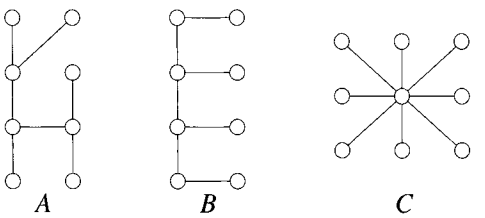


Fig. 4.1. Three trees

Adding one edge for the one that was deleted, we find that the number of edges in G is

$$(v_1 - 1) + (v_2 - 1) + 1 = v - 1.$$

(ii) Conversely, suppose G is not a tree. Select an edge that is *not* a bridge, and delete it. If the resulting graph is not a tree, repeat the process. Eventually there will be only bridges left, and the graph is a tree. From what we have just said it must have $v - 1$ edges, and the original graph had more than $v - 1$ edges. □

The word "leaf" is used to refer to a vertex of degree 1 in a tree, together with the edge incident with it.

Corollary 4.3. *Every tree other than K_1 has at least two leaves.*

Proof. Suppose the tree has v vertices. It then has $v - 1$ edges. So, by Theorem 1.1, the sum of all degrees of the vertices is $2(v - 1)$. There can be no vertex of degree 0, since the tree is connected; if $v - 1$ of the vertices have degree at least 2, then the sum of the degrees is at least $1 + 2(v - 1)$, which is impossible. □

The corollary does not hold if we allow our graphs to have infinite vertex-sets. One elementary example consists of the infinitely many vertices $0, 1, 2, \ldots, n, \ldots$ and the edges $01, 12, 23, \ldots, (n, n + 1), \ldots$. The only vertex with degree 1 is vertex 0; every other vertex in the "tree" has degree 2.

The following interesting theorem uses Corollary 4.3.

Theorem 4.4. *Suppose T is a tree with k edges and G is a graph with minimum degree $\delta(G) \geq k$. Then G has a subgraph isomorphic to T.*

Proof. The proof uses induction on k. If $k = 0$, then $T = K_1$, which is a subgraph of every graph. Suppose $k > 0$, and suppose the theorem is true for all nonnegative integers less than k. Select a vertex x of degree 1 in T (the existence of such a vertex is guaranteed by Corollary 4.3). Say wx is the edge of T containing x.

The graph $T - x$ is a tree with $k - 1$ edges, so it is isomorphic to some subgraph H of G (since $\delta(G) \geq k > k - 1$). Suppose y is the vertex of G corresponding to w. Since y has degree at least k in G, and H contains only $K - 1$ edges, there must be at least one edge adjacent to y, say yz, which is not an edge of H. Then $H + yz$ is isomorphic to T. □

Exercises 4.1

4.1.1 Show that there are exactly six nonisomorphic trees on six vertices.

4.1.2 Prove Theorem 4.1.

A4.1.3 Prove that a finite graph on v vertices that contains no cycle is connected if and only if it has $v - 1$ edges.

4.1.4 Prove that a connected graph is a tree if and only if it has the following property: *if x and y are distinct vertices, then there is a unique path in G from x to y.*

A4.1.5 A perfect square was defined in Exercise 2.1.11. Prove that no tree other than K_1 or K_2 is a perfect square.

A4.1.6 Give an example of an infinite "tree" that contains no vertex of degree 1.

H4.1.7 Let T be a tree on v vertices, $v \geq 5$, with precisely four vertices of degree 1 each and precisely one vertex of degree 4. Find the degrees of the remaining vertices of T, and show that T can be written as the union of two edge-disjoint simple walks.

H4.1.8 Let the vertices of a tree T be labeled with the integers 1, 2, ..., v. As usual, $D(i, j)$ denotes the distance between vertices i and j. Let M_T be the $v \times v$ matrix with (i, j) entry $x^{D(i,j)}$. Show that the determinant of M_T equals $(1 - x^2)^{n-1}$.

A4.1.9 A tree T with v vertices has a vertex of degree k. Prove that the longest path in T has at most $v - k + 1$ edges.

4.1.10 Prove that a graph is a tree if and only if every vertex of degree greater than 1 is a cutpoint.

4.1.11 The *center* $C(G)$ of a finite graph G of radius R was defined in Section 2.2 to consist of all those vertices x that have eccentricity $\varepsilon(x) = R$.

H(i) Prove that the center of a tree consists of either one vertex or two adjacent vertices. [73]

(ii) Give examples of trees with centers of size 1 and size 2.

4.1.12 Recall that a graph G is called *self-centered* if $C(G) = G$. Which trees are self-centered?

4.1.13 Let A be the incidence matrix of a tree on t vertices. Consider the t rows of A as vectors over GF[2], by interpreting 0 and 1 as the elements of the two-element field GF[2]. Show that any $t - 1$ rows of A are linearly independent over GF[2].

4.2 Spanning Trees

Recall that a subgraph of a graph G *spans* G if it contains every vertex of G. A *spanning tree* is a spanning subgraph that is a tree when considered as a graph in its own right.

Theorem 4.5. *Every connected graph G has a spanning tree.*

Proof. If G is a tree, then the whole of G is itself the spanning tree. Otherwise G contains a cycle. Let a be an edge in the cycle. Then a is not a bridge in G, so the graph G' obtained by deleting a from G is still connected. We have not deleted any vertex, so G' is a spanning subgraph. If G' contains a cycle, we delete an edge from that cycle. The new graph we obtain is again a connected spanning subgraph of G. This process may be continued until the remaining graph contains no cycle — that is, it is a tree. So, when the process stops, we have found a spanning tree. But the process must stop since G is finite and there are only finitely many edges that could be deleted. □

It is easy to see that Theorem 4.5 generalizes to graphs with loops and multiple edges.

It is clear from the above proof that a given multigraph may have many different spanning trees. In certain applications it is useful to know the exact number. We shall write $\tau(G)$ for the *number of spanning trees* of a graph G.

One can sometimes calculate $\tau(G)$ quite quickly. If G is a tree, then $\tau(G) = 1$. If G is a cycle of length n, then n spanning trees can be constructed, each by deleting one edge, so $\tau(G) = n$. We can consider a general multigraph G: the existence of loops does not change $\tau(G)$, as no loop can contribute to a tree; if one edge is multiple, of multiplicity k, then each spanning tree includes at most one of the k edges, and replacing one edge joining the two vertices by another gives another spanning tree.

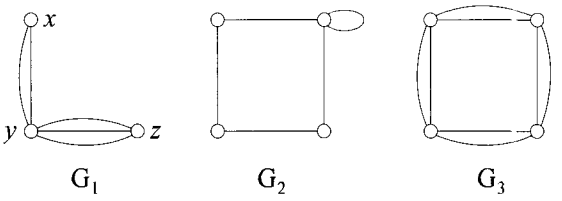


Fig. 4.2. Multigraphs whose trees are to be counted

Example. Figure 4.2 shows multigraphs G_1, G_2, G_3. We find the number of spanning trees for each.

A spanning tree in G_1 must contain one of the edges xy and one of the edges yz. The number of choices is $2 \times 3 = 6$. So $\tau(G_1) = 6$. Since loops do not affect the function τ, $\tau(G_2) = \tau(C_4) = 4$. To calculate $\tau(G_3)$, one first observes that three pairs of vertices must be joined. This can be done in four ways. In each case there are eight trees. So $\tau(G_3) = 4 \times 8 = 32$.

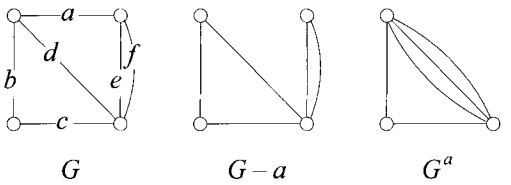


Fig. 4.3. The multigraphs used in counting trees

Calculation of $\tau(G)$ by counting becomes very tedious when G is large. In order to derive a formula for $\tau(G)$, we first introduce a new multigraph G_a. If a is any edge of a multigraph G, G_a is formed by identifying the endpoints of a: if $a = xy$, then G_a is formed by deleting both x and y, inserting a new vertex, and replacing every edge zx and every edge zy by an edge from z to the new vertex. For convenience we assume that every edge from x to y is deleted; alternatively, we could introduce these edges as loops in G_a, but this is unnecessary, as G_a will be used only in counting trees, and loops are immaterial in that context. We also use the multigraph $G - a$, formed from G by deleting a. Examples of G_a and $G - a$ are shown in Figure 4.3.

Suppose a is an edge of G. Then the spanning trees of G either contain a or they do not. A spanning tree that does not contain a is a subgraph of $G - a$, and is still a spanning subgraph, so it is a spanning tree of $G - a$; conversely, the spanning trees of $G - a$ are spanning trees of G and do not contain a. So the two sets, the spanning trees of $G - a$ and the spanning trees of G that do not contain a, are in one-to-one correspondence. So the sets are equal in size, and there are $\tau(G - a)$ spanning trees of G that do not contain a. Similarly, the number of spanning trees of G that do contain a is $\tau(G_a)$; Exercise 4.2.1 asks for a proof of this fact, but we look first at the special case shown in Figure 4.3. The spanning trees of G that contain a must also contain *either* b and c only, *or* one of b and c together with one of d, e, and f. Exactly the same is true of the spanning trees of G_a: they contain b and c only, *or* one of b and c together with one of d, e, and f.

Thus, summing the number of spanning trees of G that do or do not contain edge a, we obtain:

Theorem 4.6.

$$\tau(G) = \tau(G - a) + \tau(G_a).$$

An n-fold path is formed from a path by replacing each edge with a multiple edge of multiplicity n. An n-fold cycle is defined similarly. These multigraphs recur frequently in applications of Theorem 4.6, so it is helpful to know their numbers of spanning trees.

Theorem 4.7. *The number of spanning trees in an n-fold path is*

$$\tau(nP_v) = n^{v-1}.$$

The number of spanning trees in an n-fold cycle is

$$\tau(nC_v) = vn^{v-1}.$$

Proof. For the multiple path, one has n choices of edge for each edge of the underlying path, giving n^{v-1} paths in all. For the multiple cycle, each spanning tree is a path; there are v choices for the pair of adjacent vertices that will not be adjacent in the spanning tree, and for each choice there are again n^{v-1} paths. □

Example. Calculate $\tau(G)$, where G is the graph of Figure 4.3.

The method of decomposing the relevant graphs is indicated in Figure 4.4 (next page).

It is clear that $\tau(G_4) = 3$ (since G_4 is a cycle), that $\tau(G_5) = 1$ (since G_5 is a tree), that $\tau(G_6) = 2$, that $\tau(G_7) = 3$ and that $\tau(G_8) = 4$. So:

$$\begin{aligned}
\tau(G_2) &= \tau(G_7) + \tau(G_8) = 3 + 4 = 7;\\
\tau(G_3) &= \tau(G_5) + \tau(G_6) = 1 + 2 = 3;\\
\tau(G_1) &= \tau(G_3) + \tau(G_4) = 3 + 3 = 6;\\
\tau(G) &= \tau(G_1) + \tau(G_2) = 6 + 7 = 13.
\end{aligned}$$

Suppose G is a graph with v vertices, T is any spanning tree in G, and a is any edge of G that is not in T. Then $T + a$ has v vertices, so it must contain a cycle. Moreover, a must be an edge in that cycle. Select an edge b of the cycle, other than a. Then $T + a - b$ will be acyclic, and it is still connected, so it is a tree.

In particular, suppose R is a spanning tree of G that has k edges in common with t, and suppose a is an edge of R (but not of T). The cycle in $T + a$ must contain an edge that is not in R, because otherwise R would contain a cycle. If such an edge is chosen as b, then the tree $T - a + b$ will have $k + 1$ edges in common with R. Call this tree T_1. One can then construct another tree T_2 that shares $k+2$ edges with R, and so on. Eventually the number of shared edges will be $v - 1$, so the tree must be R. We have proved:

Theorem 4.8. *If T and R are spanning trees of the v-vertex graph G, then there exists a sequence of spanning trees,*

$$T = T_0, T_1, \ldots, T_n = R,$$

where T_i and T_{i+1} have $v - 2$ common edges for every i.

Exercises 4.2

4.2.1 Prove that there is a one-to-one correspondence between the trees of G containing edge a and the trees of G_a.

4.2.2 A multigraph G consists of a multigraph H, together with one new vertex x and an edge from x to one of the vertices of H. Show that $\tau(G) = \tau(H)$.

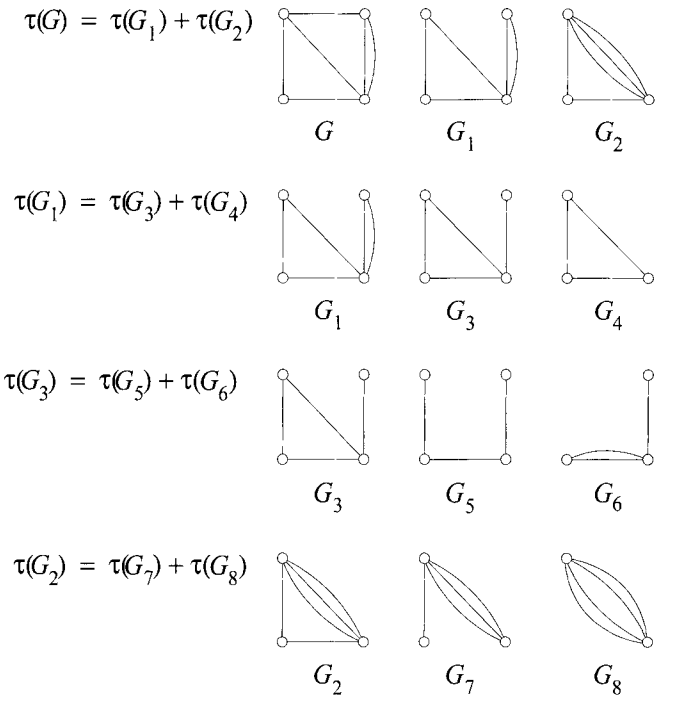


Fig. 4.4. Counting trees

A4.2.3 Show that K_v contains a pair of edge-disjoint spanning trees if and only if $v \geq 4$.

A4.2.4 Find the number of spanning trees in each of the graphs and multigraphs shown in Figure 4.5.

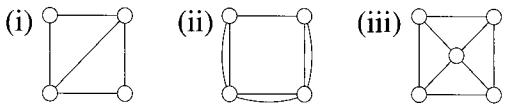


Fig. 4.5. Count the spanning trees

4.2.5 Repeat the preceding exercise for the graphs and multigraphs shown in Figure 4.6.

4.2.6 Recall that a graph is called cubic if every vertex has degree 3.

(i) Prove that if a cubic graph on n vertices contains two edge-disjoint spanning trees, then $n \leq 8$.

(ii) Is there a cubic graph on four vertices containing two edge-disjoint spanning trees? Is there one which does not contain two edge-disjoint spanning trees?

(iii) Repeat part (ii) for $n = 6$ and for $n = 8$.

A4.2.7 Let G be the graph with four vertices 1, 2, 3, 4 and two edges (1, 2), (3, 4). Construct multigraphs N_1, N_2, N_3, N_4, with the following properties: each N_i consists of four edges, and four, five or six vertices; each N_i contains G as a subgraph; $\tau(N_i) = i - 1$, for $i = 1, 2, 3, 4$.

4.2.8 Prove that every bridge in a connected graph lies on every spanning tree of the graph.

A4.2.9 Find $\tau(K_4)$ and $\tau(K_5)$.

H4.2.10 Suppose a graph G is formed by taking two disjoint connected graphs G_1 and G_2 and identifying a vertex in G_1 with a vertex in G_2. Show that $\tau(G) = \tau(G_1)\tau(G_2)$.

4.2.11 Suppose G is formed by taking two disjoint connected graphs G_1 and G_2 and inserting an edge connecting some vertex of G_1 with some vertex of G_2. Use Theorem 4.5 and the result of the preceding Exercise to find an expression for $\tau(G)$.

4.2.12 Let T_1 and T_2 be spanning trees of a connected graph G; show that if a is any edge of T_1, then there exists an edge b of T_2 such that $(T_1 - \{a\}) \cup \{b\}$ (the graph obtained from T_1 on replacing a by b) is also a spanning tree. Show also that T_1 can be "transformed" into T_2 by replacing the edges of T_1 one at a time by edges of T_2 in such a way that at each stage we obtain a spanning tree.

4.2.13 (i) Show that in any connected graph, any cycle must have at least one edge in common with the complement of any spanning tree.

(ii) Show that in any connected graph, any cutset must have at least one edge in common with any spanning tree.

A4.2.14 Let H be a subgraph of a connected graph G. Show that H is a subgraph of some spanning tree T of G if and only if H contains no cycle.

4.2.15 If G is any connected graph or multigraph with v vertices, the *tree graph* of G has as its vertices the spanning trees of G; two vertices are adjacent if and only if the

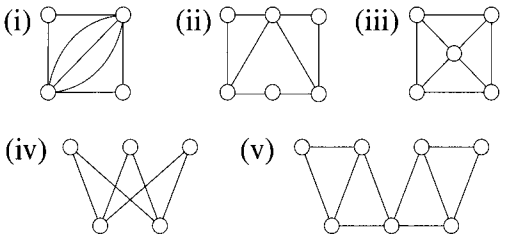


Fig. 4.6. Count the spanning trees

trees have $v - 2$ edges in common. Prove that the tree graph of a graph is always connected.

4.3 Minimal Spanning Trees

Consider applications of the kind discussed in Section 2.3, where each edge of a graph has a weight associated with it. It is sometimes desirable to find a spanning tree such that the weight of the tree — the total of the weights of its edges — is minimum. Such a tree is called a *minimal spanning tree.*

It is clear that a finite graph can contain only finitely many spanning trees, so it is possible in theory to list all spanning trees and their weights, and to find a minimal spanning tree by choosing one with minimum weight. This process could take a very long time however, since $\tau(G)$ can be very large. So efficient algorithms that find a minimal spanning tree are useful. We present here an example due to Prim [96].

We assume that G is a graph with vertex-set V and edge set E, and suppose there is associated with G a map $w\colon E \to R$ called the *weight* of the edge; when xy is an edge of G we write $w(x, y)$ for the image of xy under w. We could quite easily modify the algorithm to allow for multiple edges, but the notation is slightly simpler in the graph case. The algorithm consists of finding a sequence of vertices $x_0, x_1, x_2, \ldots$, of G and a sequence of sets $S_0, S_1, S_2, \ldots$, where

$$S_i = \{x_0, x_1, \ldots, x_{i-1}\}.$$

We choose x_0 at random from V. When $n > 0$, we find x_n inductively using S_n as follows.

1. Given i, $0 \leq i \leq n-1$, choose y_i to be a member of $V \setminus S_n$ such that $w(x_i, y_i)$ is minimum, if possible. In other words:
   a) if there is no member of $V \setminus S_n$ adjacent to x_i, then there is no y_i;
   b) if $V \setminus S_n$ contains a vertex adjacent to x_i, then y_i is one of those vertices adjacent to x_i, and if $x_i \sim y$, then $w(x_i, y_i) \leq w(x_i, y)$.
2. Provided that at least one y_i has been found in Step (1), then define x_n to be a y_i such that $w(x_i, y_i)$ is minimal; in other words, x_n is the y_i that satisfies

$$w(x_i, y_i) \leq w(x_j, y_j) \text{ for all } j.$$

3. Put $S_{n+1} = S_n \cup \{x_n\}$.

This process stops only when there is no new vertex y_i. If there is no new vertex y_i, it must be true that no member of $V \setminus S_n$ is adjacent to a vertex of S_n. It is impossible to partition the vertices of a connected graph into two nonempty sets such that no edge joins one set to the other, and S_n is never empty, so the process stops only when $S_n = V$.

When we reach this stage, so that $S_n = V$, we construct a graph T as follows:
(i) T has vertex-set V;
(ii) if x_k arose as y_i, then x_i is adjacent to x_k in T;
(iii) no edges of T exist other than those that may be found using (ii).
It is not hard to verify that T is a tree and that it is minimal; see Exercise 4.3.3.

Observe that xX_n may not be defined uniquely at Step (2) of the algorithm, and indeed y_i may not be uniquely defined. This is to be expected: after all, there may be more than one minimal spanning tree.

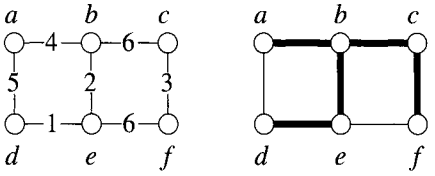


Fig. 4.7. An example of Prim's algorithm

Example. Consider the graph G shown in Figure 4.7. Weights are shown next to the edges.

Select $x_0 = a$. Then $S_1 = \{a\}$. Now $y_0 = b$, and this is the only choice for x_1. So $S_2 = \{a, b\}$. The tree will contain edge ab.

Working from S_2, we get $y_0 = d$ and $y_1 = e$. Since be $(= x_1y_1)$ has smaller weight than ad $(= x_0y_0)$, we select $x_2 = e$. Then $S_3 = \{a, b, e\}$ and edge be goes into the tree. Similarly, from S_3, we get $x_3 = d$ and $S_4 = \{a, b, d, e\}$, and the new edge is de.

Now there is a choice. Working from S_4, $y_1 = c$ and $y_2 = f$. In both cases the weight of the edge to be considered is 6. So either may be used. Let us choose c, and use edge bc.

The final vertex is f, and the edge is cf. So the tree has edges ab, bc, be, cf, de and weight 16.

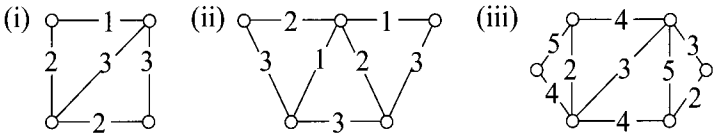
The algorithm might be described as follows. First, choose a vertex x_0. Trivially the minimum weight tree with vertex-set $\{x_0\}$ — the *only* tree with vertex-set $\{x_0\}$ — is the K_1 with vertex x_0. Call this the *champion*. Then find the smallest weight tree with two vertices, one of which is x_0; in other words, find the minimum weight tree that can be formed by adding just one edge to the current champion. This tree is the new champion. Continue in this way: each time a champion is found, look for the cheapest tree that can be formed by adding one edge to it. One can consider each new tree to be

an approximation to the final minimal spanning tree, with successive approximations having more and more edges.

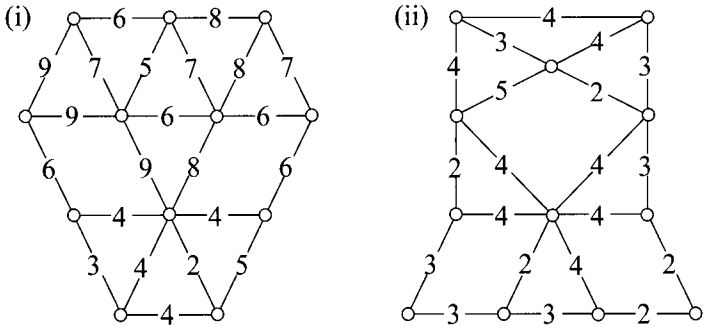
Prim's algorithm was a refinement of an earlier algorithm due to Kruskal [81]. In that algorithm, one starts by listing all edges in order of increasing weight. The first approximation is the K_2 consisting of the edge of least weight. The second approximation is formed by appending the next edge in the ordering. At each stage the next approximation is formed by adding on the smallest edge that has not been used, provided only that it does not form a cycle with the edges already chosen. In this case the successive approximations are not necessarily connected, until the last one. The advantage of Prim's algorithm is that, in large graphs, the initial sorting stage of Kruskal's algorithm can be very time consuming.

Exercises 4.3

4.3.1 Find minimal spanning trees in the following graphs, using both Kruskal's and Prim's methods.

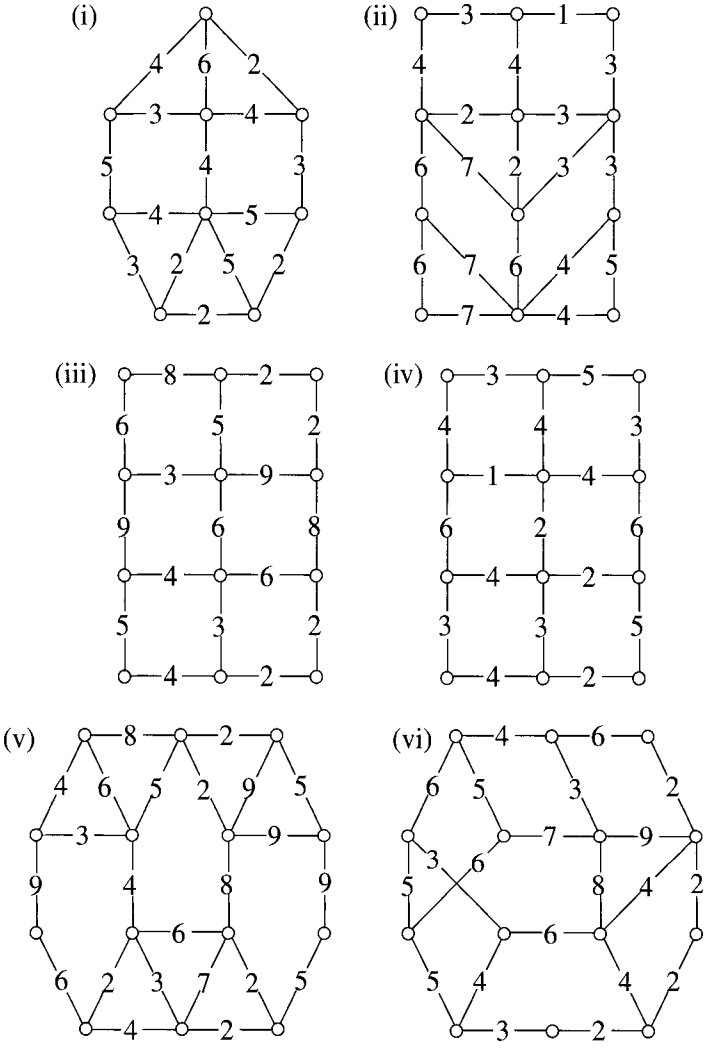


A4.3.2 Find minimal spanning trees in the following graphs, using both Kruskal's and Prim's methods.



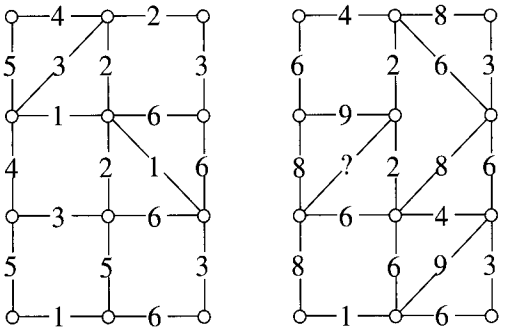
4.3.3 Prove that the graph T constructed in Prim's algorithm *is* in fact a minimal spanning tree.

4.3.4 Find minimal spanning trees in the following graphs, using both Kruskal's and Prim's methods.



H4.3.5 It is required to find a *maximal* spanning tree in a graph. Suggest a modification of Prim's algorithm for this problem.

A4.3.6 (i) On graph A below, a weight function is shown. Find a minimal spanning tree in A.



A *B*

(ii) *B* is similar, except that no weight is specified for one edge. Find a minimal spanning tree in *B* if that edge has weight

(a) 1; (b) 4; (c) 7.